|  |  |
| --- | --- |
|  | ***Patuakhali Science and Technology University*** |

Assignment on

***“*Solve Exercise**”

Course Code: CCE-122

Course Title: Object Oriented Programming

Level - I; Semester - II

|  |
| --- |
| **Submitted By**  **Name: M.D. Sakibul Islam Shovon**  **ID:** 2302056  **REG:** 11834  **Session:** 2023-2024  Faculty of Computer Science and Engineering |

|  |
| --- |
| **Submitted To**  **Prof. Dr. Md. Samsuzzaman**  Professor of Computer and Communication Engineering Department  Faculty of Computer Science and Engineering |

**2. Also solve the below exercise**

Section 2.2  
▼2.2.1  
Identify and fix the errors in the following code:  
  
 1  public class Test {  
 2    public void main(string[] args) {  
 3      double i = 50.0;  
 4      double k = i + 50.0;  
 5      double j = k + 1;  
 6  
 7      System.out.println("j is " + j + " and  
 8        k is " + k);  
 9    }  
10  }

Answer**:**

Line 2: string should be String (capital S) and main method should be static

Line 7-8: String literal is incorrectly split across lines

Fixed code:

public class Test {

public static void main(String[] args) {

double i = 50.0;

double k = i + 50.0;

double j = k + 1;

System.out.println("j is " + j + " and k is " + k);

}

}

Section 2.3  
▼2.3.1  
How do you write a statement to let the user enter a double value from the keyboard? What happens if you entered 5a when executing the following code?  
  
double radius = input.nextDouble();

Answer**:**

The user enter a double value from the keyboard:

double value = input.nextDouble();  
If we input 5a in double radius = input.nextDouble(); this statement  it throws an InputMismatchException because "5a" is not a valid double.

▼2.3.2  
Are there any performance differences between the following two import statements?  
  
import java.util.Scanner;  
import java.util.\*;  
Answer:

Both compile to the same bytecode.

import java.util.\* may slow compilation slightly but runs the same as import java.util.Scanner;.

Section 2.4  
▼2.4.1  
Which of the following identifiers are valid? Which are Java keywords?  
  
miles, Test, a++, --a, 4#R, $4, #44, apps  
class, public, int, x, y, radius

Answer:

Valid identifiers: miles, Test, $4, apps, x, y, radius

Java keywords: class, public, int

Invalid identifiers: a++ (contains +), --a (contains --), 4#R (starts with digit), #44 (contains #)

Section 2.5  
▼2.5.1  
Identify and fix the errors in the following code:  
  
1  public class Test {  
2    public static void main(String[] args) {  
3      int i = k + 2;  
4      System.out.println(i);  
5    }  
6  }

Answer:

Variable k is used before declaration (line 3).

public class Test {

public static void main(String[] args) {

int k = 0;

int i = k + 2;

System.out.println(i);

}

}

Section 2.6  
▼2.6.1  
Identify and fix the errors in the following code:  
  
1  public class Test {  
2    public static void main(String[] args) {  
3      int i = j = k = 2;  
4      System.out.println(i + " " + j + " " + k);  
5    }  
6  }

Answer:

Multiple variables (j, k) are not declared before being assigned in the same line.

public class Test {

public static void main(String[] args) {

int i, j, k;

i = j = k = 2;

System.out.println(i + " " + j + " " + k);

}

}

Section 2.7  
▼2.7.1  
What are the benefits of using constants? Declare an int constant SIZE with value 20.

Answer:

Readability – Names (e.g., SIZE) clarify meaning vs. magic numbers (e.g., 20).

Maintainability – Change value once (in declaration) instead of multiple places.

Safety – Prevents accidental modification (compile-time error if reassigned).

Declare SIZE as a constant:

public static final int SIZE = 20;

▼2.7.2  
Translate the following algorithm into Java code:  
Step 1: Declare a double variable named miles with initial value 100.  
Step 2: Declare a double constant named KILOMETERS\_PER\_MILE with value 1.609.  
Step 3: Declare a double variable named kilometers, multiply miles and KILOMETERS\_PER\_MILE, and assign the result to kilometers.  
Step 4: Display kilometers to the console.  
What is kilometers after Step 4?

Answer:

public class Main {

public static void main(String[] args) {

double miles = 100;

final double KILOMETERS\_PER\_MILE = 1.609;

double kilometers = miles \* KILOMETERS\_PER\_MILE;

System.out.println(kilometers);

}

}

Result after Step 4: kilometers = 160.9

Section 2.8  
▼2.8.1  
What are the naming conventions for class names, method names, constants, and variables? Which of the following items can be a constant, a method, a variable, or a class according to the Java naming conventions?  
  
MAX\_VALUE, Test, read, readDouble

Answer:

Naming Conventions:

* Class: PascalCase (First letter of *each word* capitalized; e.g., Test)
* Method: camelCase (First letter *lowercase*, subsequent words capitalized;e.g., readDouble)
* Variable: camelCase (First letter *lowercase*, subsequent words capitalized;e.g., count)
* Constant: UPPER\_SNAKE\_CASE (*All uppercase* with underscores between words;e.g., MAX\_VALUE)

Given Examples:

* MAX\_VALUE → Constant
* Test → Class
* read → Method or Variable
* readDouble → Method

Section 2.9  
▼2.9.1  
Find the largest and smallest byte, short, int, long, float, and double. Which of these data types requires the least amount of memory?

Answer:

**Largest and Smallest Values:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Type** | **Minimum Value** | **Maximum Value** | **Memory (Bytes)** |
| byte | -128 | 127 | **1** |
| short | -32,768 | 32,767 | **2** |
| int | -2³¹ (-2,147,483,648) | 2³¹-1 (2,147,483,647) | **4** |
| long | -2⁶³ | 2⁶³-1 | **8** |
| float | ±1.4E-45 (approx.) | ±3.4E+38 (approx.) | **4** |
| double | ±4.9E-324 (approx.) | ±1.8E+308 (approx.) | **8** |

The data types requires the least amount of memory byte (1 byte)

▼2.9.2  
Show the result of the following remainders.  
  
56 % 6  
78 % -4  
-34 % 5  
-34 % -5  
5 % 1  
1 % 5

Answer:

56 % 6 → **2**

78 % -4 → **2**

-34 % 5 → **-4**

-34 % -5 → **-4**

5 % 1 → **0**

1 % 5 → **1**

▼2.9.3  
If today is Tuesday, what will be the day in 100 days?

Answer:

Week has 7 days → 100 % 7 = 2 (remainder)

Tuesday + 2 days = **Thursday**

▼2.9.4  
What is the result of 25 / 4? How would you rewrite the expression if you wished the result to be a floating-point number?

Answer:

Result of 25 / 4 = 6, because integer division

Floating-point result Rewrite as : 25.0 / 4

Result: 6.25, because floating division

▼2.9.5  
Show the result of the following code:  
  
System.out.println(2 \* (5 / 2 + 5 / 2));  
System.out.println(2 \* 5 / 2 + 2 \* 5 / 2);  
System.out.println(2 \* (5 / 2));  
System.out.println(2 \* 5 / 2);

Answer:

System.out.println(2 \* (5 / 2 + 5 / 2)); → 8

System.out.println(2 \* 5 / 2 + 2 \* 5 / 2); → 10

System.out.println(2 \* (5 / 2)); → 4

System.out.println(2 \* 5 / 2); → 5

▼2.9.6  
Are the following statements correct? If so, show the output.  
  
System.out.println("25 / 4 is " + 25 / 4);  
System.out.println("25 / 4.0 is " + 25 / 4.0);  
System.out.println("3 \* 2 / 4 is " + 3 \* 2 / 4);  
System.out.println("3.0 \* 2 / 4 is " + 3.0 \* 2 / 4);

Answer:

System.out.println("25 / 4 is " + 25 / 4); → "25 / 4 is 6"

System.out.println("25 / 4.0 is " + 25 / 4.0); → "25 / 4.0 is 6.25"

System.out.println("3 \* 2 / 4 is " + 3 \* 2 / 4); → "3 \* 2 / 4 is 1"

System.out.println("3.0 \* 2 / 4 is " + 3.0 \* 2 / 4); → "3.0 \* 2 / 4 is 1.5"

▼2.9.7  
Write a statement to display the result of 2 3.5 .

Answer:

The correct statement to display the result of 2 \* 3.5 in Java is:

System.out.println(2 \* 3.5);

Output:  
7.0

▼2.9.8  
Suppose m and r are integers. Write a Java expression for mr 2 to obtain a floating-point result.

Answer:

To compute  (mr²) as a **floating-point result** in Java when m and r are integers, use:

(double) m \* r \* r

▼2.10.1  
How many accurate digits are stored in a float or double type variable?

Answer:

Float (32-bit): ~6-7 accurate decimal digits.  
Double (64-bit): ~15-16 accurate decimal digits.

▼2.10.2  
Which of the following are correct literals for floating-point numbers?  
  
12.3, 12.3e+2, 23.4e-2, -334.4, 20.5, 39F, 40D

Answer:

Correct Floating-Point Literals:

1. 12.3 (double)
2. 12.3e+2 (scientific notation, double)
3. 23.4e-2 (scientific notation, double)
4. -334.4 (double, negative)
5. 20.5 (double)
6. 39F (float, suffix F)
7. 40D (double, suffix D)

▼2.10.3  
Which of the following are the same as 52.534?  
  
5.2534e+1, 0.52534e+2, 525.34e-1, 5.2534e+0

Answer:

5.2534e+1 : 5.2534×101=52.5345.2534×101=52.534 → Same

0.52534e+2 : 0.52534×102=52.5340.52534×102=52.534 → Same

525.34e-1 : 525.34×10−1=52.534525.34×10−1=52.534 → Same

5.2534e+0 : 5.2534×100=5.25345.2534×100=5.2534 → Not the same

▼2.10.4  
Which of the following are correct literals?  
  
5\_2534e+1, \_2534, 5\_2, 5\_

Answer:

Correct Literals:

1. 5\_2534e+1: Valid (underscores in numeric literals are allowed, even in scientific notation). Equivalent to 52534.0 (double).
2. 5\_2: Valid (underscore in integer literal). Equivalent to 52 (int).

Incorrect Literals:

1. \_2534Invalid (underscore at the start is not allowed). Correct form: 2534 or 2\_534.
2. 5\_: Invalid (underscore at the end is not allowed). Correct form: 5.

Underscores (\_) are allowed between digits for readability (e.g., 1\_00\_000).

▼2.11.1  
How would you write the following arithmetic expression in Java?  
a.  
b. 5.5 \* (r + 2.5) 2.5 + t

Answe:

a) No expression here.

b) There are missing \* operator between (r+2.5) and 2.5.

▼2.12.1  
How do you obtain the current second, minute, and hour?

Answer:

In Java, you can obtain the current second, minute, and hour using the java.time.LocalTime or java.time.LocalDateTime classes (modern approach) or the older java.util.Calendar class.

import java.time.LocalTime;

public class CurrentTime {

public static void main(String[] args) {

LocalTime now = LocalTime.now();

int hour = now.getHour();

int minute = now.getMinute();

int second = now.getSecond();

System.out.println("Current Time: " + hour + ":" + minute + ":" + second);

}

}

Show the output of the following code:  
  
double a = 6.5;  
a += a + 1;  
System.out.println(a);  
a = 6;  
a /= 2;  
System.out.println(a);

Answer:

Output: 14.0

Section 2.14  
▼2.14.1  
Which of these statements are true?  
a. Any expression can be used as a statement.  
b. The expression x++ can be used as a statement.  
c. The statement x = x + 5 is also an expression.  
d. The statement x = y = x = 0 is illegal.

Answer:

a. True – Any expression can be used as a statement when followed by a semicolon (e.g., x++;, Math.pow(2, 3);).

b. True – x++ is an expression and can be used as a statement (e.g., x++;).

c. True – x = x + 5 is both a statement and an expression (it evaluates to the assigned value).

d. False – x = y = x = 0 is legal (assigns 0 to x, y, and x again, right-to-left).

▼2.14.2  
Show the output of the following code:  
  
int a = 6;  
int b = a++;  
System.out.println(a);  
System.out.println(b);  
a = 6;  
b = ++a;  
System.out.println(a);  
System.out.println(b);

Answer:

Output:

7

6

7

7

Section 2.15  
▼2.15.1  
Can different types of numeric values be used together in a computation?

Answer:

Yes, different types of numeric values can be used together in a computation in Java. When you perform operations with mixed numeric types, Java follows implicit type conversion (promotion) rules to ensure compatibility

▼2.15.2  
What does an explicit casting from a double to an int do with the fractional part of the double value? Does casting change the variable being cast?

Answer:

1. Effect on the Fractional Part:
   * When you explicitly cast a double to an int, Java truncates (discards) the fractional part (no rounding occurs).
   * Example:

double d = 9.99;

int i = (int) d; *// i becomes 9 (0.99 is lost)*

System.out.println(i); *// Output: 9*

1. Does Casting Change the Original Variable?
   * No, casting does not modify the original variable. It only converts the value temporarily for the assignment or operation.
   * Example:

double d = 5.7;

int i = (int) d; *// i = 5, but d remains 5.7*

System.out.println(d); *// Output: 5.7 (unchanged)*

▼2.15.3  
Show the output of the following code:  
  
float f = 12.5F;  
int i = (int)f;  
System.out.println("f is " + f);  
System.out.println("i is " + i);

Answer:

f is 12.5

i is 12

▼2.15.4  
If you change (int)(tax \* 100) / 100.0 to (int)(tax \* 100) / 100 in line 11 in Listing 2.8, what will be the output for the input purchase amount of 197.556?

Answer:

Original Code:

java

double tax = (int)(tax \* 100) / 100.0;

(int)(tax \* 100) truncates to an integer (e.g., 19755 for 197.556).

Division by 100.0 preserves the decimal (result: 197.55).

Modified Code:

double tax = (int)(tax \* 100) / 100;

(int)(tax \* 100) truncates (e.g., 19755 for 197.556).

Division by 100 (integer division) discards the fractional part (result: 197.0).

Output for Input 197.556:

Original Output (/ 100.0): tax = 197.55 (correctly rounded to 2 decimal places).

Modified Output (/ 100): tax = 197.0 (incorrect, loses all cents due to integer division).

▼2.15.5  
Show the output of the following code:  
  
double amount = 5;  
System.out.println(amount / 2);  
System.out.println(5 / 2);

Answer:

Output:

2.5

2

▼2.15.6  
Write an expression that rounds up a double value in variable d to an integer.

Answer;

To round **up** a double value stored in variable d to the nearest integer.

Section 2.16  
▼2.16.1  
How would you write the following arithmetic expression?

Answer:

**Arithmetic Examples**

|  |  |
| --- | --- |
| Mathematical Expression | Java Code |
| a+bc−d*c*−*da*+*b*​ | (a + b) / (c - d) |
| 3x2+5x−23*x*2+5*x*−2 | 3 \* x \* x + 5 \* x - 2 |
| b2−4ac*b*2−4*ac*​ | Math.sqrt(b \* b - 4 \* a \* c) |
| 59(F−32)95​(*F*−32) (Fahrenheit to Celsius) | (5.0 / 9) \* (F - 32) |

▼2.17.1  
Show the output in Listing 2.10 with the input value 1.99.

Answer:

▼2.18.1  
Can you declare a variable as int and later redeclare it as double?

Answer:

No, you cannot redeclare a variable with a different type in the same scope. Once a variable is declared as int, it cannot be redeclared as double later in the same block.

▼2.18.2  
What is an integer overflow? Can floating-point operations cause overflow?

Answer:

Integer overflow occurs when an arithmetic operation on integers produces a result outside the range of the data type (e.g., exceeding Integer.MAX\_VALUE).

Floating-point operations can also overflow, but the behavior differs:

Overflow: Results in Infinity (positive or negative).

Underflow: Results in 0.0 (for values too small to represent).

▼2.18.3  
Will overflow cause a runtime error?

Answer:

Integer Overflow:  No runtime error (silent wrap-around).

Floating-Point Overflow:  No runtime error (results in Infinity or 0.0).

▼2.18.4  
What is a round-off error? Can integer operations cause round-off errors? Can floating-point operations cause round-off errors?

Answer:

A round-off error occurs when a numerical value cannot be represented exactly due to limitations in data type precision, leading to small inaccuracies in calculations.

Pure integer arithmetic (e.g., int, long) does not suffer from round-off errors because:

- Integers are stored exactly in binary.

-Operations like +, -, \* preserve exactness (unless they cause overflow).

Floating arithmetic (e.g., int, long) does not suffer from round-off errors because

-Floating-point types (float, double) use binary fractions, which cannot precisely

represent all decimal numbers (e.g., 0.1 in decimal is an infinite repeating fraction in

binary).